FOUNDATION-0.2b: Agent State Tables - PART 2 (Testing)

🎯 CONTEXT

Phase: FOUNDATION (Week 1 - Day 1 Afternoon)
Component: Agent State & Configuration Tables - Testing & Validation
Estimated Time: 15 min AI execution + 10 min verification
Complexity: MEDIUM
Risk Level: LOW
Files: Part 2 of 2 (Testing, seed data, validation)

📋 PREREQUISITES

PART 1 Must Be Complete:

✅ All 4 models created (`agent_state.py`)
✅ Migration file created (`002_agent_state_tables.py`)
✅ Models imported in `__init__.py`
✅ Migration executed (`alembic upgrade head`)
✅ 4 new tables exist in PostgreSQL

Verify PART 1 Completion:

bash

```
# Check tables exist
psql postgresql://optiinfra:password@localhost:5432/optiinfra -c "\dt" | grep agent

# Expected output:
#  agent_configs
#  agent_states
#  agent_capabilities
#  agent_metrics

# Check migration status
cd ~/optiinfra/shared/database
alembic current

# Expected: 002_agent_state_tables (head)
```

If any checks fail, complete PART 1 first!

📁 FILE 1: Seed Data Script

Location: `~/optiinfra/shared/database/scripts/seed_agent_state.py`

python

```
"""
Seed data for agent state tables (FOUNDATION-0.2b)
Populates: agent_configs, agent_states, agent_capabilities, agent_metrics
"""

from datetime import datetime, timedelta
from sqlalchemy.orm import Session
from shared.database.models import (
    Agent, AgentConfig, AgentState, AgentCapability, AgentMetric
)
from shared.database.session import SessionLocal

def seed_agent_state_tables(db: Session):
    """Seed agent state tables with test data"""

    print("🌱 Seeding agent state tables...")

    # Get existing test agents from FOUNDATION-0.2a
    cost_agent = db.query(Agent).filter_by(type="cost", name="Cost Optimizer").first()
    perf_agent = db.query(Agent).filter_by(type="performance", name="Performance Tuner").first()
    resource_agent = db.query(Agent).filter_by(type="resource", name="Resource Allocator").first()
    app_agent = db.query(Agent).filter_by(type="application", name="Application Monitor").first()

    if not all([cost_agent, perf_agent, resource_agent, app_agent]):
        raise ValueError("❌ Core agents not found! Run FOUNDATION-0.2a seed data first.")

    # ========================================
    # SEED AGENT CONFIGS
    # ========================================
    print("  📋 Seeding agent_configs...")

    configs = [
        # Cost Agent Config
        AgentConfig(
            agent_id=cost_agent.id,
            config_key="cost_threshold_usd",
            config_value="1000",
            config_type="number",
            description="Minimum monthly cost to trigger optimization",
            is_sensitive=False
        ),
AgentConfig(
    agent_id=cost_agent.id,
    config_key="savings_threshold_percent",
    config_value="15",
    config_type="number",
    description="Minimum savings percentage to recommend",
    is_sensitive=False
),
AgentConfig(
    agent_id=cost_agent.id,
    config_key="check_interval_minutes",
    config_value="60",
    config_type="number",
    description="How often to check for cost anomalies",
    is_sensitive=False
),

# Performance Agent Config
AgentConfig(
    agent_id=perf_agent.id,
    config_key="latency_p95_threshold_ms",
    config_value="500",
    config_type="number",
    description="P95 latency threshold for alerts",
    is_sensitive=False
),
AgentConfig(
    agent_id=perf_agent.id,
    config_key="throughput_threshold_rps",
    config_value="100",
    config_type="number",
    description="Minimum throughput (requests/sec)",
    is_sensitive=False
),
AgentConfig(
    agent_id=perf_agent.id,
    config_key="kv_cache_size_gb",
    config_value="32",
    config_type="number",
    description="KV cache size in GB",
    is_sensitive=False
```

```
),

# Resource Agent Config
AgentConfig(
    agent_id=resource_agent.id,
    config_key="gpu_utilization_target",
    config_value="75",
    config_type="number",
    description="Target GPU utilization percentage",
    is_sensitive=False
),
AgentConfig(
    agent_id=resource_agent.id,
    config_key="scale_up_threshold",
    config_value="85",
    config_type="number",
    description="GPU utilization % to trigger scale-up",
    is_sensitive=False
),
AgentConfig(
    agent_id=resource_agent.id,
    config_key="scale_down_threshold",
    config_value="40",
    config_type="number",
    description="GPU utilization % to trigger scale-down",
    is_sensitive=False
),

# Application Agent Config
AgentConfig(
    agent_id=app_agent.id,
    config_key="quality_score_threshold",
    config_value="0.85",
    config_type="number",
    description="Minimum quality score (0-1)",
    is_sensitive=False
),
AgentConfig(
    agent_id=app_agent.id,
    config_key="hallucination_threshold",
    config_value="0.05",
        config_type="number",
        description="Maximum hallucination rate",
        is_sensitive=False
    ),
    AgentConfig(
        agent_id=app_agent.id,
        config_key="regression_check_interval",
        config_value="15",
        config_type="number",
        description="Minutes between regression checks",
        is_sensitive=False
    ),
]

db.add_all(configs)
db.commit()
print(f"   ✅ Created {len(configs)} agent configs")

# ============================================
# SEED AGENT STATES
# ============================================
print("  🔄 Seeding agent_states...")

now = datetime.utcnow()

states = [
    # Cost Agent - Currently analyzing
    AgentState(
        agent_id=cost_agent.id,
        status="active",
        current_workflow="cost_analysis",
        workflow_data={
            "customer_id": "550e8400-e29b-41d4-a716-446655440000",
            "analysis_started": now.isoformat(),
            "metrics_collected": 150,
            "savings_identified": 45000
        },
        resource_locks=["gpu_cluster_1", "billing_data"],
        error_count=0,
        last_error=None,
        last_heartbeat=now
```

```
),

# Performance Agent - Idle but healthy
AgentState(
    agent_id=perf_agent.id,
    status="idle",
    current_workflow=None,
    workflow_data={},
    resource_locks=[],
    error_count=0,
    last_error=None,
    last_heartbeat=now
),

# Resource Agent - Active with workflow
AgentState(
    agent_id=resource_agent.id,
    status="active",
    current_workflow="gpu_scaling",
    workflow_data={
        "customer_id": "550e8400-e29b-41d4-a716-446655440000",
        "current_gpus": 8,
        "target_gpus": 12,
        "scale_reason": "high_utilization",
        "started_at": now.isoformat()
    },
    resource_locks=["k8s_cluster", "gpu_pool"],
    error_count=1,
    last_error={
        "timestamp": (now - timedelta(hours=2)).isoformat(),
        "error": "Transient network timeout",
        "recovered": True
    },
    last_heartbeat=now
),

# Application Agent - Error state (for testing)
AgentState(
    agent_id=app_agent.id,
    status="error",
    current_workflow="quality_check",
        workflow_data={
            "customer_id": "550e8400-e29b-41d4-a716-446655440000",
            "check_id": "qc_20251019_001",
            "samples_checked": 50,
            "failures": 3
        },
        resource_locks=["quality_db"],
        error_count=3,
        last_error={
            "timestamp": (now - timedelta(minutes=5)).isoformat(),
            "error": "Quality threshold breach: score=0.82 (threshold=0.85)",
            "severity": "high"
        },
        last_heartbeat=now - timedelta(minutes=5)
    ),
]

db.add_all(states)
db.commit()
print(f"   ✅ Created {len(states)} agent states")

# ============================================
# SEED AGENT CAPABILITIES
# ============================================
print("  💪 Seeding agent_capabilities...")

capabilities = [
    # Cost Agent Capabilities
    AgentCapability(
        agent_id=cost_agent.id,
        capability_name="gpu_cost_analysis",
        capability_version="1.2.0",
        description="Analyze GPU cluster costs and identify savings",
        parameters={
            "supported_providers": ["aws", "gcp", "azure"],
            "gpu_types": ["H100", "A100", "V100"],
            "analysis_depth": "detailed"
        },
        is_enabled=True
    ),
    AgentCapability(
    agent_id=cost_agent.id,
    capability_name="storage_optimization",
    capability_version="1.0.0",
    description="Optimize storage tiering and costs",
    parameters={
        "storage_types": ["s3", "ebs", "gcs"],
        "tiering_policies": ["hot", "warm", "cold"]
    },
    is_enabled=True
),
AgentCapability(
    agent_id=cost_agent.id,
    capability_name="spot_instance_recommendation",
    capability_version="0.9.0",
    description="Recommend spot instance usage (beta)",
    parameters={
        "max_spot_percentage": 70,
        "fallback_strategy": "on_demand"
    },
    is_enabled=False  # Beta - disabled
),

# Performance Agent Capabilities
AgentCapability(
    agent_id=perf_agent.id,
    capability_name="kv_cache_tuning",
    capability_version="2.1.0",
    description="Optimize KV cache settings for vLLM/TGI",
    parameters={
        "supported_engines": ["vllm", "tgi", "sglang"],
        "cache_strategies": ["static", "dynamic", "adaptive"]
    },
    is_enabled=True
),
AgentCapability(
    agent_id=perf_agent.id,
    capability_name="batch_optimization",
    capability_version="1.5.0",
    description="Optimize batch sizes and continuous batching",
    parameters={
        "min_batch_size": 4,
        "max_batch_size": 128,
        "dynamic_batching": True
    },
    is_enabled=True
),
AgentCapability(
    agent_id=perf_agent.id,
    capability_name="quantization_testing",
    capability_version="1.0.0",
    description="Test FP8/INT8 quantization strategies",
    parameters={
        "quantization_formats": ["fp8", "int8", "int4"],
        "quality_validation": True
    },
    is_enabled=True
),

# Resource Agent Capabilities
AgentCapability(
    agent_id=resource_agent.id,
    capability_name="auto_scaling",
    capability_version="2.0.0",
    description="Auto-scale GPU clusters based on demand",
    parameters={
        "scaling_triggers": ["utilization", "queue_depth", "latency"],
        "cooldown_minutes": 10,
        "max_scale_factor": 3
    },
    is_enabled=True
),
AgentCapability(
    agent_id=resource_agent.id,
    capability_name="load_balancing",
    capability_version="1.3.0",
    description="Balance workloads across GPU clusters",
    parameters={
        "strategy": "least_loaded",
        "affinity_rules": True
    },
    is_enabled=True
),
    # Application Agent Capabilities
    AgentCapability(
        agent_id=app_agent.id,
        capability_name="quality_monitoring",
        capability_version="1.4.0",
        description="Monitor LLM output quality in real-time",
        parameters={
            "metrics": ["coherence", "relevance", "factuality"],
            "sampling_rate": 0.1,
            "alert_threshold": 0.85
        },
        is_enabled=True
    ),
    AgentCapability(
        agent_id=app_agent.id,
        capability_name="regression_detection",
        capability_version="1.1.0",
        description="Detect quality regressions after changes",
        parameters={
            "baseline_samples": 100,
            "comparison_samples": 100,
            "significance_level": 0.05
        },
        is_enabled=True
    ),
    AgentCapability(
        agent_id=app_agent.id,
        capability_name="hallucination_detection",
        capability_version="0.8.0",
        description="Detect hallucinations in LLM outputs (experimental)",
        parameters={
            "detection_methods": ["fact_checking", "consistency"],
            "confidence_threshold": 0.7
        },
        is_enabled=False  # Experimental
    ),
]

db.add_all(capabilities)
db.commit()
```

```python
print(f"   ✅ Created {len(capabilities)} agent capabilities")

# ==========================================
# SEED AGENT METRICS
# ==========================================
print("  📊 Seeding agent_metrics...")

metrics = [
    # Cost Agent Metrics
    AgentMetric(
        agent_id=cost_agent.id,
        metric_name="total_savings_identified_usd",
        metric_value=125000.00,
        metric_unit="usd",
        collection_timestamp=now,
        metadata={
            "period": "last_30_days",
            "customers_analyzed": 5,
            "recommendations_generated": 12
        }
    ),
    AgentMetric(
        agent_id=cost_agent.id,
        metric_name="average_savings_percent",
        metric_value=42.5,
        metric_unit="percent",
        collection_timestamp=now,
        metadata={
            "min": 28.0,
            "max": 67.0,
            "median": 41.0
        }
    ),
    AgentMetric(
        agent_id=cost_agent.id,
        metric_name="analysis_execution_time_seconds",
        metric_value=145.3,
        metric_unit="seconds",
        collection_timestamp=now,
        metadata={
            "p50": 120.0,
            "p95": 180.0,
            "p99": 220.0
        }
    ),

    # Performance Agent Metrics
    AgentMetric(
        agent_id=perf_agent.id,
        metric_name="latency_improvement_percent",
        metric_value=65.0,
        metric_unit="percent",
        collection_timestamp=now,
        metadata={
            "before_p95_ms": 800,
            "after_p95_ms": 280,
            "optimizations_applied": 8
        }
    ),
    AgentMetric(
        agent_id=perf_agent.id,
        metric_name="throughput_increase_percent",
        metric_value=180.0,
        metric_unit="percent",
        collection_timestamp=now,
        metadata={
            "before_rps": 50,
            "after_rps": 140,
            "batch_size_optimized": True
        }
    ),
    AgentMetric(
        agent_id=perf_agent.id,
        metric_name="tuning_execution_time_seconds",
        metric_value=320.5,
        metric_unit="seconds",
        collection_timestamp=now,
        metadata={
            "tests_run": 25,
            "configurations_tested": 15
        }
    ),
    # Resource Agent Metrics
    AgentMetric(
        agent_id=resource_agent.id,
        metric_name="average_gpu_utilization_percent",
        metric_value=78.5,
        metric_unit="percent",
        collection_timestamp=now,
        metadata={
            "before": 35.0,
            "after": 78.5,
            "improvement": 124.3
        }
    ),
    AgentMetric(
        agent_id=resource_agent.id,
        metric_name="scaling_decisions_count",
        metric_value=45,
        metric_unit="count",
        collection_timestamp=now,
        metadata={
            "scale_ups": 23,
            "scale_downs": 22,
            "period_days": 30
        }
    ),
    AgentMetric(
        agent_id=resource_agent.id,
        metric_name="scaling_accuracy_percent",
        metric_value=92.0,
        metric_unit="percent",
        collection_timestamp=now,
        metadata={
            "correct_decisions": 41,
            "total_decisions": 45,
            "false_positives": 4
        }
    ),

    # Application Agent Metrics
    AgentMetric(
            agent_id=app_agent.id,
            metric_name="quality_score_average",
            metric_value=0.91,
            metric_unit="score",
            collection_timestamp=now,
            metadata={
                "samples_checked": 1500,
                "period": "last_7_days",
                "threshold": 0.85
            }
        ),
        AgentMetric(
            agent_id=app_agent.id,
            metric_name="regressions_detected_count",
            metric_value=3,
            metric_unit="count",
            collection_timestamp=now,
            metadata={
                "caught_before_production": 3,
                "auto_rolled_back": 2,
                "manual_review": 1
            }
        ),
        AgentMetric(
            agent_id=app_agent.id,
            metric_name="check_execution_time_seconds",
            metric_value=28.7,
            metric_unit="seconds",
            collection_timestamp=now,
            metadata={
                "samples_per_check": 100,
                "parallel_checks": 4
            }
        ),
    ]

    db.add_all(metrics)
    db.commit()
    print(f"   ✅ Created {len(metrics)} agent metrics")

    print("\n✅ Agent state tables seeded successfully!")
```

```python
    print(f"   📋 Configs: {len(configs)}")
    print(f"   🔄 States: {len(states)}")
    print(f"   💪 Capabilities: {len(capabilities)}")
    print(f"   📊 Metrics: {len(metrics)}")

def main():
    """Main entry point"""
    db = SessionLocal()
    try:
        seed_agent_state_tables(db)
    except Exception as e:
        print(f"❌ Error seeding agent state tables: {e}")
        db.rollback()
        raise
    finally:
        db.close()

if __name__ == "__main__":
    main()
```

🧪 FILE 2: Test Fixtures

Location: `~/optiinfra/shared/database/tests/conftest.py` (append to existing)

python

```python
# Add these fixtures to existing conftest.py

import pytest
from datetime import datetime, timedelta
from shared.database.models import (
    AgentConfig, AgentState, AgentCapability, AgentMetric
)

@pytest.fixture
def sample_agent_config(db_session, sample_cost_agent):
    """Create sample agent config"""
    config = AgentConfig(
        agent_id=sample_cost_agent.id,
        config_key="test_threshold",
        config_value="100",
        config_type="number",
        description="Test configuration",
        is_sensitive=False
    )
    db_session.add(config)
    db_session.commit()
    db_session.refresh(config)
    return config

@pytest.fixture
def sample_agent_state(db_session, sample_cost_agent):
    """Create sample agent state"""
    state = AgentState(
        agent_id=sample_cost_agent.id,
        status="active",
        current_workflow="test_workflow",
        workflow_data={"key": "value"},
        resource_locks=["resource1"],
        error_count=0,
        last_error=None,
        last_heartbeat=datetime.utcnow()
    )
    db_session.add(state)
    db_session.commit()
    db_session.refresh(state)
    return state
```

```
@pytest.fixture
def sample_agent_capability(db_session, sample_cost_agent):
    """Create sample agent capability"""
    capability = AgentCapability(
        agent_id=sample_cost_agent.id,
        capability_name="test_capability",
        capability_version="1.0.0",
        description="Test capability",
        parameters={"param1": "value1"},
        is_enabled=True
    )
    db_session.add(capability)
    db_session.commit()
    db_session.refresh(capability)
    return capability

@pytest.fixture
def sample_agent_metric(db_session, sample_cost_agent):
    """Create sample agent metric"""
    metric = AgentMetric(
        agent_id=sample_cost_agent.id,
        metric_name="test_metric",
        metric_value=100.0,
        metric_unit="count",
        collection_timestamp=datetime.utcnow(),
        metadata={"source": "test"}
    )
    db_session.add(metric)
    db_session.commit()
    db_session.refresh(metric)
    return metric
```

🧪 FILE 3: Comprehensive Test Suite

Location: `~/optiinfra/shared/database/tests/test_agent_state_models.py`

python

```python
"""
Test suite for agent state tables (FOUNDATION-0.2b)
Tests: AgentConfig, AgentState, AgentCapability, AgentMetric
"""

import pytest
from datetime import datetime, timedelta
from sqlalchemy.exc import IntegrityError
from shared.database.models import (
    Agent, AgentConfig, AgentState, AgentCapability, AgentMetric
)

class TestAgentConfig:
    """Test AgentConfig model and relationships"""

    def test_create_agent_config(self, db_session, sample_cost_agent):
        """Test creating an agent config"""
        config = AgentConfig(
            agent_id=sample_cost_agent.id,
            config_key="max_cost_threshold",
            config_value="5000",
            config_type="number",
            description="Maximum cost threshold",
            is_sensitive=False
        )
        db_session.add(config)
        db_session.commit()

        assert config.id is not None
        assert config.config_key == "max_cost_threshold"
        assert config.config_value == "5000"
        assert config.config_type == "number"
        assert config.is_sensitive is False
        assert config.created_at is not None
        assert config.updated_at is not None

    def test_agent_config_relationship(self, db_session, sample_agent_config):
        """Test agent -> configs relationship"""
        agent = db_session.query(Agent).filter_by(
            id=sample_agent_config.agent_id
        ).first()
        assert len(agent.configs) >= 1
        assert sample_agent_config in agent.configs
        assert agent.configs[0].agent_id == agent.id

    def test_agent_config_foreign_key(self, db_session):
        """Test foreign key constraint"""
        config = AgentConfig(
            agent_id=99999,  # Non-existent agent
            config_key="test",
            config_value="test",
            config_type="string",
            description="Test"
        )
        db_session.add(config)

        with pytest.raises(IntegrityError):
            db_session.commit()

    def test_agent_config_sensitive_flag(self, db_session, sample_cost_agent):
        """Test sensitive configuration handling"""
        sensitive_config = AgentConfig(
            agent_id=sample_cost_agent.id,
            config_key="api_key",
            config_value="secret_key_123",
            config_type="string",
            description="API Key",
            is_sensitive=True
        )
        db_session.add(sensitive_config)
        db_session.commit()

        # Retrieve and verify
        retrieved = db_session.query(AgentState).filter_by(id=state.id).first()
        assert retrieved.workflow_data["analysis"]["total_cost"] == 120000
        assert len(retrieved.workflow_data["analysis"]["recommendations"]) == 2

class TestAgentCapability:
    """Test AgentCapability model and versioning"""

    def test_create_agent_capability(self, db_session, sample_cost_agent):
        """Test creating an agent capability"""
        capability = AgentCapability(
            agent_id=sample_cost_agent.id,
            capability_name="advanced_cost_analysis",
            capability_version="2.1.0",
            description="Advanced cost analysis with ML",
            parameters={
                "model": "cost_predictor_v2",
                "confidence_threshold": 0.85,
                "features": ["gpu_usage", "network", "storage"]
            },
            is_enabled=True
        )
        db_session.add(capability)
        db_session.commit()

        assert capability.id is not None
        assert capability.capability_name == "advanced_cost_analysis"
        assert capability.capability_version == "2.1.0"
        assert capability.is_enabled is True
        assert "model" in capability.parameters
        assert len(capability.parameters["features"]) == 3

    def test_agent_capability_relationship(self, db_session, sample_agent_capability):
        """Test agent -> capabilities relationship"""
        agent = db_session.query(Agent).filter_by(
            id=sample_agent_capability.agent_id
        ).first()

        assert len(agent.capabilities) >= 1
        assert sample_agent_capability in agent.capabilities

    def test_multiple_capability_versions(self, db_session, sample_cost_agent):
        """Test storing multiple versions of same capability"""
        # Version 1.0
        cap_v1 = AgentCapability(
            agent_id=sample_cost_agent.id,
            capability_name="cost_optimization",
            capability_version="1.0.0",
            description="Basic cost optimization",
            parameters={"method": "basic"},
            is_enabled=False
        )

        # Version 2.0
        cap_v2 = AgentCapability(
            agent_id=sample_cost_agent.id,
            capability_name="cost_optimization",
            capability_version="2.0.0",
            description="Advanced cost optimization",
            parameters={"method": "advanced", "ml_enabled": True},
            is_enabled=True
        )

        db_session.add_all([cap_v1, cap_v2])
        db_session.commit()

        # Query both versions
        capabilities = db_session.query(AgentCapability).filter_by(
            agent_id=sample_cost_agent.id,
            capability_name="cost_optimization"
        ).all()

        assert len(capabilities) == 2
        versions = [c.capability_version for c in capabilities]
        assert "1.0.0" in versions
        assert "2.0.0" in versions

    def test_capability_enable_disable(self, db_session, sample_agent_capability):
        """Test enabling/disabling capabilities"""
        # Initially enabled
        assert sample_agent_capability.is_enabled is True

        # Disable
        sample_agent_capability.is_enabled = False
        db_session.commit()

        # Verify
        retrieved = db_session.query(AgentCapability).filter_by(
            id=sample_agent_capability.id
        ).first()
        assert retrieved.is_enabled is False
```

```python
    def test_capability_parameters_update(self, db_session, sample_agent_capability):
        """Test updating capability parameters"""
        # Update parameters
        sample_agent_capability.parameters = {
            "new_param": "value",
            "threshold": 0.95
        }
        db_session.commit()

        # Verify
        retrieved = db_session.query(AgentCapability).filter_by(
            id=sample_agent_capability.id
        ).first()
        assert "new_param" in retrieved.parameters
        assert retrieved.parameters["threshold"] == 0.95

class TestAgentMetric:
    """Test AgentMetric model and time-series data"""

    def test_create_agent_metric(self, db_session, sample_cost_agent):
        """Test creating an agent metric"""
        now = datetime.utcnow()
        metric = AgentMetric(
            agent_id=sample_cost_agent.id,
            metric_name="cost_savings_usd",
            metric_value=45000.50,
            metric_unit="usd",
            collection_timestamp=now,
            metadata={
                "period": "monthly",
                "customer_count": 5
            }
        )
        db_session.add(metric)
        db_session.commit()

        assert metric.id is not None
        assert metric.metric_name == "cost_savings_usd"
        assert metric.metric_value == 45000.50
        assert metric.metric_unit == "usd"
        assert metric.collection_timestamp == now
        assert metric.metadata["period"] == "monthly"

    def test_agent_metric_relationship(self, db_session, sample_agent_metric):
        """Test agent -> metrics relationship"""
        agent = db_session.query(Agent).filter_by(
            id=sample_agent_metric.agent_id
        ).first()

        assert len(agent.metrics) >= 1
        assert sample_agent_metric in agent.metrics

    def test_metric_time_series(self, db_session, sample_cost_agent):
        """Test storing time-series metrics"""
        base_time = datetime.utcnow()

        # Create hourly metrics
        metrics = []
        for hour in range(5):
            metric = AgentMetric(
                agent_id=sample_cost_agent.id,
                metric_name="gpu_utilization",
                metric_value=70.0 + hour * 2,
                metric_unit="percent",
                collection_timestamp=base_time + timedelta(hours=hour),
                metadata={"hour": hour}
            )
            metrics.append(metric)

        db_session.add_all(metrics)
        db_session.commit()

        # Query time range
        start_time = base_time
        end_time = base_time + timedelta(hours=5)

        retrieved = db_session.query(AgentMetric).filter(
            AgentMetric.agent_id == sample_cost_agent.id,
            AgentMetric.metric_name == "gpu_utilization",
            AgentMetric.collection_timestamp >= start_time,
            AgentMetric.collection_timestamp <= end_time
        ).order_by(AgentMetric.collection_timestamp).all()

        assert len(retrieved) == 5
        assert retrieved[0].metric_value == 70.0
        assert retrieved[4].metric_value == 78.0

    def test_metric_aggregation(self, db_session, sample_cost_agent):
        """Test metric aggregation queries"""
        from sqlalchemy import func

        # Create multiple metrics
        values = [100.0, 200.0, 150.0, 175.0, 225.0]
        for value in values:
            metric = AgentMetric(
                agent_id=sample_cost_agent.id,
                metric_name="request_count",
                metric_value=value,
                metric_unit="count",
                collection_timestamp=datetime.utcnow(),
                metadata={}
            )
            db_session.add(metric)
        db_session.commit()

        # Calculate average
        avg_value = db_session.query(
            func.avg(AgentMetric.metric_value)
        ).filter(
            AgentMetric.agent_id == sample_cost_agent.id,
            AgentMetric.metric_name == "request_count"
        ).scalar()

        assert avg_value == 170.0  # (100+200+150+175+225)/5

    def test_metric_metadata_search(self, db_session, sample_cost_agent):
        """Test searching metrics by metadata"""
        # Create metrics with different metadata
        metric1 = AgentMetric(
            agent_id=sample_cost_agent.id,
            metric_name="latency",
            metric_value=100.0,
            metric_unit="ms",
            collection_timestamp=datetime.utcnow(),
            metadata={"region": "us-west", "env": "prod"}
        )
        metric2 = AgentMetric(
            agent_id=sample_cost_agent.id,
            metric_name="latency",
            metric_value=150.0,
            metric_unit="ms",
            collection_timestamp=datetime.utcnow(),
            metadata={"region": "us-east", "env": "prod"}
        )
        db_session.add_all([metric1, metric2])
        db_session.commit()

        # Query by metadata (PostgreSQL JSONB query)
        result = db_session.query(AgentMetric).filter(
            AgentMetric.agent_id == sample_cost_agent.id,
            AgentMetric.metadata["region"].astext == "us-west"
        ).first()

        assert result is not None
        assert result.metric_value == 100.0

class TestAgentStateIntegration:
    """Integration tests across all agent state tables"""

    def test_complete_agent_with_all_state_tables(
        self, db_session, sample_cost_agent
    ):
        """Test agent with all state table relationships"""
        # Create config
        config = AgentConfig(
            agent_id=sample_cost_agent.id,
            config_key="threshold",
            config_value="100",
            config_type="number",
            description="Test threshold"
        )

        # Create state
```

```python
state = AgentState(
    agent_id=sample_cost_agent.id,
    status="active",
    current_workflow="test",
    workflow_data={},
    resource_locks=[],
    error_count=0,
    last_heartbeat=datetime.utcnow()
)

# Create capability
capability = AgentCapability(
    agent_id=sample_cost_agent.id,
    capability_name="test_cap",
    capability_version="1.0.0",
    description="Test",
    parameters={},
    is_enabled=True
)

# Create metric
metric = AgentMetric(
    agent_id=sample_cost_agent.id,
    metric_name="test_metric",
    metric_value=100.0,
    metric_unit="count",
    collection_timestamp=datetime.utcnow(),
    metadata={}
)

db_session.add_all([config, state, capability, metric])
db_session.commit()

# Verify all relationships
agent = db_session.query(Agent).filter_by(
    id=sample_cost_agent.id
).first()

assert len(agent.configs) >= 1
assert agent.state is not None
assert len(agent.capabilities) >= 1
```

```
        assert len(agent.metrics) >= 1

    def test_agent_deletion_cascades(self, db_session):
        """Test that deleting agent cascades to all state tables"""
        # Create agent
        agent = Agent(
            type="test",
            name="Test Agent",
            version="1.0.0",
            status="active",
            endpoint="http://test:8000",
            capabilities=["test"],
            last_heartbeat=datetime.utcnow()
        )
        db_session.add(agent)
        db_session.commit()
        agent_id = agent.id

        # Create related records
        config = AgentConfig(
            agent_id=agent_id,
            config_key="test",
            config_value="test",
            config_type="string",
            description="Test"
        )
        state = AgentState(
            agent_id=agent_id,
            status="active",
            current_workflow=None,
            workflow_data={},
            resource_locks=[],
            error_count=0,
            last_heartbeat=datetime.utcnow()
        )
        capability = AgentCapability(
            agent_id=agent_id,
            capability_name="test",
            capability_version="1.0.0",
            description="Test",
            parameters={},
            is_enabled=True
        )
        metric = AgentMetric(
            agent_id=agent_id,
            metric_name="test",
            metric_value=1.0,
            metric_unit="count",
            collection_timestamp=datetime.utcnow(),
            metadata={}
        )

        db_session.add_all([config, state, capability, metric])
        db_session.commit()

        # Store IDs
        config_id = config.id
        state_id = state.id
        capability_id = capability.id
        metric_id = metric.id

        # Delete agent
        db_session.delete(agent)
        db_session.commit()

        # Verify all related records deleted (CASCADE)
        assert db_session.query(Agent).filter_by(id=agent_id).first() is None
        assert db_session.query(AgentConfig).filter_by(id=config_id).first() is None
        assert db_session.query(AgentState).filter_by(id=state_id).first() is None
        assert db_session.query(AgentCapability).filter_by(id=capability_id).first() is None
        assert db_session.query(AgentMetric).filter_by(id=metric_id).first() is None

    def test_query_agent_with_active_workflow(self, db_session):
        """Test querying agents with active workflows"""
        # Create multiple agents with different states
        agent1 = Agent(
            type="cost", name="Agent 1", version="1.0.0",
            status="active", endpoint="http://a1:8000",
            capabilities=["test"], last_heartbeat=datetime.utcnow()
        )
        agent2 = Agent(
            type="performance", name="Agent 2", version="1.0.0",
    status="active", endpoint="http://a2:8000",
    capabilities=["test"], last_heartbeat=datetime.utcnow()
)
db_session.add_all([agent1, agent2])
db_session.commit()

# Agent1 has active workflow
state1 = AgentState(
    agent_id=agent1.id,
    status="active",
    current_workflow="analysis",
    workflow_data={},
    resource_locks=["resource1"],
    error_count=0,
    last_heartbeat=datetime.utcnow()
)

# Agent2 is idle
state2 = AgentState(
    agent_id=agent2.id,
    status="idle",
    current_workflow=None,
    workflow_data={},
    resource_locks=[],
    error_count=0,
    last_heartbeat=datetime.utcnow()
)

db_session.add_all([state1, state2])
db_session.commit()

# Query agents with active workflows
active_agents = db_session.query(Agent).join(AgentState).filter(
    AgentState.status == "active",
    AgentState.current_workflow.isnot(None)
).all()

assert len(active_agents) == 1
assert active_agents[0].id == agent1.id
```

🚀 EXECUTION STEPS

Step 1: Create Seed Data Script

bash

```bash
cd ~/optiinfra/shared/database

# Create scripts directory if it doesn't exist
mkdir -p scripts

# Create seed script
cat > scripts/seed_agent_state.py << 'EOF'
[Copy the seed_agent_state.py content from FILE 1 above]
EOF

# Make executable
chmod +x scripts/seed_agent_state.py
```

Step 2: Add Test Fixtures

bash

```bash
cd ~/optiinfra/shared/database/tests

# Append fixtures to conftest.py
cat >> conftest.py << 'EOF'

# ============================================
# FOUNDATION-0.2b FIXTURES
# ============================================
[Copy the fixture content from FILE 2 above]
EOF
```

Step 3: Create Test File

bash

```bash
cd ~/optiinfra/shared/database/tests

# Create test file
cat > test_agent_state_models.py << 'EOF'
[Copy the test content from FILE 3 above]
EOF
```

Step 4: Run Seed Data

bash

```bash
cd ~/optiinfra/shared/database

# Run seed script
python scripts/seed_agent_state.py

# Expected output:
# 🌱 Seeding agent state tables...
#   📋 Seeding agent_configs...
#     ✅ Created 12 agent configs
#   🔄 Seeding agent_states...
#     ✅ Created 4 agent states
#   💪 Seeding agent_capabilities...
#     ✅ Created 11 agent capabilities
#   📊 Seeding agent_metrics...
#     ✅ Created 12 agent metrics
#
# ✅ Agent state tables seeded successfully!
#    📋 Configs: 12
#    🔄 States: 4
#    💪 Capabilities: 11
#    📊 Metrics: 12
```

Step 5: Run Tests

bash

```
cd ~/optiinfra/shared/database

# Run all agent state tests
pytest tests/test_agent_state_models.py -v

# Expected output:
# test_agent_state_models.py::TestAgentConfig::test_create_agent_config PASSED
# test_agent_state_models.py::TestAgentConfig::test_agent_config_relationship PASSED
# test_agent_state_models.py::TestAgentConfig::test_agent_config_foreign_key PASSED
# test_agent_state_models.py::TestAgentConfig::test_agent_config_sensitive_flag PASSED
# test_agent_state_models.py::TestAgentConfig::test_agent_config_cascade_delete PASSED
# test_agent_state_models.py::TestAgentState::test_create_agent_state PASSED
# test_agent_state_models.py::TestAgentState::test_agent_state_relationship PASSED
# test_agent_state_models.py::TestAgentState::test_agent_state_error_tracking PASSED
# test_agent_state_models.py::TestAgentState::test_agent_state_heartbeat_tracking PASSED
# test_agent_state_models.py::TestAgentState::test_agent_state_workflow_data PASSED
# test_agent_state_models.py::TestAgentCapability::test_create_agent_capability PASSED
# test_agent_state_models.py::TestAgentCapability::test_agent_capability_relationship PASSED
# test_agent_state_models.py::TestAgentCapability::test_multiple_capability_versions PASSED
# test_agent_state_models.py::TestAgentCapability::test_capability_enable_disable PASSED
# test_agent_state_models.py::TestAgentCapability::test_capability_parameters_update PASSED
# test_agent_state_models.py::TestAgentMetric::test_create_agent_metric PASSED
# test_agent_state_models.py::TestAgentMetric::test_agent_metric_relationship PASSED
# test_agent_state_models.py::TestAgentMetric::test_metric_time_series PASSED
# test_agent_state_models.py::TestAgentMetric::test_metric_aggregation PASSED
# test_agent_state_models.py::TestAgentMetric::test_metric_metadata_search PASSED
# test_agent_state_models.py::TestAgentStateIntegration::test_complete_agent_with_all_state_tables PASSED
# test_agent_state_models.py::TestAgentStateIntegration::test_agent_deletion_cascades PASSED
# test_agent_state_models.py::TestAgentStateIntegration::test_query_agent_with_active_workflow PASSED
#
# ======================== 23 passed in 2.15s ========================

# Run with coverage
pytest tests/test_agent_state_models.py --cov=shared.database.models.agent_state --cov-report=term-missing

# Expected coverage: >95%
```

✅ VALIDATION CHECKLIST

Database Verification

bash

```
# Connect to PostgreSQL
psql postgresql://optiinfra:password@localhost:5432/optiinfra

# Check table counts
SELECT 'agent_configs' as table_name, COUNT(*) FROM agent_configs
UNION ALL
SELECT 'agent_states', COUNT(*) FROM agent_states
UNION ALL
SELECT 'agent_capabilities', COUNT(*) FROM agent_capabilities
UNION ALL
SELECT 'agent_metrics', COUNT(*) FROM agent_metrics;

# Expected:
#  agent_configs      | 12
#  agent_states       | 4
#  agent_capabilities | 11
#  agent_metrics      | 12

# Verify relationships
SELECT
    a.name as agent_name,
    COUNT(DISTINCT c.id) as configs,
    COUNT(DISTINCT s.id) as states,
    COUNT(DISTINCT cap.id) as capabilities,
    COUNT(DISTINCT m.id) as metrics
FROM agents a
LEFT JOIN agent_configs c ON a.id = c.agent_id
LEFT JOIN agent_states s ON a.id = s.agent_id
LEFT JOIN agent_capabilities cap ON a.id = cap.agent_id
LEFT JOIN agent_metrics m ON a.id = m.agent_id
GROUP BY a.name;

# Expected: Each agent should have configs, state, capabilities, metrics

# Check specific data
SELECT
    config_key,
    config_value,
    config_type
FROM agent_configs
WHERE agent_id = (SELECT id FROM agents WHERE type = 'cost' LIMIT 1);
```

```
# Should show cost agent configs

\q
```

Python Verification

bash

```
cd ~/optiinfra

# Test imports
python << 'EOF'
from shared.database.models import (
    AgentConfig, AgentState, AgentCapability, AgentMetric
)
from shared.database.session import SessionLocal

db = SessionLocal()

# Count records
print(f"Configs: {db.query(AgentConfig).count()}")
print(f"States: {db.query(AgentState).count()}")
print(f"Capabilities: {db.query(AgentCapability).count()}")
print(f"Metrics: {db.query(AgentMetric).count()}")

# Test relationships
from shared.database.models import Agent
agent = db.query(Agent).filter_by(type="cost").first()
print(f"\nCost Agent '{agent.name}':")
print(f"  - Configs: {len(agent.configs)}")
print(f"  - State: {agent.state.status if agent.state else 'None'}")
print(f"  - Capabilities: {len(agent.capabilities)}")
print(f"  - Metrics: {len(agent.metrics)}")

db.close()
print("\n✅ All verifications passed!")
EOF

# Expected output:
# Configs: 12
# States: 4
# Capabilities: 11
# Metrics: 12
#
# Cost Agent 'Cost Optimizer':
#   - Configs: 3
#   - State: active
#   - Capabilities: 3
#   - Metrics: 3
```

```
#
# ✅ All verifications passed!
```

🔧 TROUBLESHOOTING

Issue 1: Seed Script Fails - "Core agents not found"

Problem: `ValueError: ❌ Core agents not found!`

Solution:

bash

```bash
# Run FOUNDATION-0.2a seed data first
cd ~/optiinfra/shared/database
python scripts/seed_database.py

# Then run 0.2b seed data
python scripts/seed_agent_state.py
```

Issue 2: Foreign Key Violation

Problem: `IntegrityError: foreign key violation`

Solution:

bash

```bash
# Check agents exist
psql postgresql://optiinfra:password@localhost:5432/optiinfra -c "SELECT id, name FROM agents;"

# If no agents, run 0.2a migration and seed:
cd ~/optiinfra/shared/database
alembic downgrade base
alembic upgrade head
python scripts/seed_database.py
python scripts/seed_agent_state.py
```

Issue 3: Tests Fail - Missing Fixtures

Problem: `NameError: fixture 'sample_cost_agent' not found`

Solution:

bash

```bash
# Verify conftest.py has core fixtures from 0.2a
cd ~/optiinfra/shared/database/tests
grep "sample_cost_agent" conftest.py

# If missing, ensure FOUNDATION-0.2a conftest.py is complete
# Then add 0.2b fixtures
```

Issue 4: Migration Already Applied

Problem: `Target database is not up to date`

Solution:

bash

```bash
cd ~/optiinfra/shared/database

# Check current version
alembic current

# If already at 002_, tables should exist
psql postgresql://optiinfra:password@localhost:5432/optiinfra -c "\dt" | grep agent

# If tables exist but seed data missing, just run seed:
python scripts/seed_agent_state.py
```

Issue 5: JSONB Query Fails

Problem: `AttributeError: 'InstrumentedAttribute' object has no attribute 'astext'`

Solution:

python

```python
# Use correct PostgreSQL JSONB syntax
from sqlalchemy.dialects.postgresql import JSONB

# Correct query:
result = db_session.query(AgentMetric).filter(
    AgentMetric.metadata['region'].astext == 'us-west'
).first()
```

📊 SUCCESS CRITERIA

✅ Must Pass (ALL required):

1. **Seed Data Loaded:**
   - ☐ 12 agent configs created
   - ☐ 4 agent states created
   - ☐ 11 agent capabilities created
   - ☐ 12 agent metrics created
2. **Tests Passing:**
   - ☐ 23/23 tests pass
   - ☐ No test errors or warnings
   - ☐ Coverage >95%
3. **Database Integrity:**
   - ☐ All foreign keys working
   - ☐ CASCADE deletes working
   - ☐ JSONB queries working
   - ☐ Indexes created
4. **Relationships:**
   - ☐ agent.configs works
   - ☐ agent.state works
   - ☐ agent.capabilities works
   - ☐ agent.metrics works
5. **Data Quality:**
   - ☐ All agents have configs
   - ☐ All agents have state
   - ☐ All agents have capabilities
   - ☐ All agents have metrics

📥 FINAL STEPS

Step 1: Verify Everything

bash

```
cd ~/optiinfra/shared/database

# Run full validation
./scripts/validate_foundation_02b.sh

# Or manually:
python scripts/seed_agent_state.py
pytest tests/test_agent_state_models.py -v --cov
```

Step 2: Git Commit

bash

```
cd ~/optiinfra

# Stage files
git add shared/database/models/agent_state.py
git add shared/database/models/core.py
git add shared/database/models/__init__.py
git add shared/database/migrations/versions/002_agent_state_tables.py
git add shared/database/scripts/seed_agent_state.py
git add shared/database/tests/conftest.py
git add shared/database/tests/test_agent_state_models.py

# Commit
git commit -m "feat(database): Add agent state tables (FOUNDATION-0.2b)

- Add AgentConfig model for agent configuration
- Add AgentState model for agent runtime state
- Add AgentCapability model for capability versioning
- Add AgentMetric model for agent metrics
- Add migration 002_agent_state_tables
- Add seed data script with 39 total records
- Add 23 comprehensive tests (all passing)
- Add relationships to core Agent model

Tables: agent_configs, agent_states, agent_capabilities, agent_metrics
Tests: 23/23 passing, >95% coverage
Seed Data: 12 configs, 4 states, 11 capabilities, 12 metrics"

# Push
git push origin main
```

🎊 PART 2 COMPLETE!

What You Achieved:

✅ **39 seed records created** across 4 tables
✅ **23 comprehensive tests** all passing
✅ **4 test fixtures** for easy testing
✅ **Complete validation** scripts and checks
✅ **Full troubleshooting** guide
✅ **Git commit** ready

System Status:

📋✓

PostgreSQL Tables: 10/10 ✅
  Core (0.2a): customers, agents, events, recommendations, approvals, optimizations
  State (0.2b): agent_configs, agent_states, agent_capabilities, agent_metrics

Seed Data: 53 records ✅
  Core: 14 records (from 0.2a)
  State: 39 records (from 0.2b)

Tests: 36/36 passing ✅
  Core: 13 tests (from 0.2a)
  State: 23 tests (from 0.2b)

Coverage: >95% ✅

⏭️ WHAT'S NEXT?

FOUNDATION-0.2b is COMPLETE! 🎉

You now have:

- Complete core database schema
- Agent state management tables
- Full test coverage
- Seed data for testing

Next FOUNDATION prompt: `FOUNDATION-0.3: ClickHouse Metrics Schema`

This will add time-series metrics storage for high-frequency data.

📥 HOW TO DOWNLOAD THIS FILE

1. Click **"Copy"** dropdown at the top
2. Select **"Download as txt"**
3. Save as: `FOUNDATION-0.2b-Agent-State-Tables-PART2-Testing.md`

✅ **FOUNDATION-0.2b COMPLETE! Ready for 0.3? 🚀** session.query(AgentConfig).filter_by(config_key="api_key").first()

```
        assert retrieved.is_sensitive is True
        assert retrieved.config_value == "secret_key_123"

    def test_agent_config_cascade_delete(self, db_session, sample_cost_agent):
        """Test CASCADE delete when agent is deleted"""
        # Create config
        config = AgentConfig(
            agent_id=sample_cost_agent.id,
            config_key="test_key",
            config_value="test_value",
            config_type="string",
            description="Test"
        )
        db_session.add(config)
        db_session.commit()
        config_id = config.id

        # Delete agent
        db_session.delete(sample_cost_agent)
        db_session.commit()

        # Config should be deleted
        assert db_session.query(AgentConfig).filter_by(id=config_id).first() is None
```

class TestAgentState: """Test AgentState model and state management"""

```
def test_create_agent_state(self, db_session, sample_cost_agent):
    """Test creating an agent state"""
    now = datetime.utcnow()
    state = AgentState(
        agent_id=sample_cost_agent.id,
        status="active",
        current_workflow="cost_analysis",
        workflow_data={"step": 1, "progress": 50},
        resource_locks=["gpu_cluster_1", "billing_data"],
        error_count=0,
        last_error=None,
        last_heartbeat=now
    )
    db_session.add(state)
    db_session.commit()

    assert state.id is not None
    assert state.status == "active"
    assert state.current_workflow == "cost_analysis"
    assert state.workflow_data["step"] == 1
    assert len(state.resource_locks) == 2
    assert "gpu_cluster_1" in state.resource_locks
    assert state.error_count == 0
    assert state.last_heartbeat == now

def test_agent_state_relationship(self, db_session, sample_agent_state):
    """Test agent -> state relationship"""
    agent = db_session.query(Agent).filter_by(
        id=sample_agent_state.agent_id
    ).first()

    assert agent.state is not None
    assert agent.state.id == sample_agent_state.id
    assert agent.state.agent_id == agent.id

def test_agent_state_error_tracking(self, db_session, sample_cost_agent):
    """Test error tracking in agent state"""
    state = AgentState(
        agent_id=sample_cost_agent.id,
        status="error",
        current_workflow="analysis",
        workflow_data={},
        resource_locks=[],
        error_count=3,
        last_error={
            "timestamp": datetime.utcnow().isoformat(),
            "error": "Connection timeout",
            "severity": "high"
        },
        last_heartbeat=datetime.utcnow()
    )
    db_session.add(state)
    db_session.commit()

    assert state.status == "error"
    assert state.error_count == 3
    assert state.last_error is not None
    assert "Connection timeout" in state.last_error["error"]

def test_agent_state_heartbeat_tracking(self, db_session, sample_agent_state):
    """Test heartbeat timestamp updates"""
    old_heartbeat = sample_agent_state.last_heartbeat

    # Simulate heartbeat update
    import time
    time.sleep(0.1)
    sample_agent_state.last_heartbeat = datetime.utcnow()
    db_session.commit()

    assert sample_agent_state.last_heartbeat > old_heartbeat

def test_agent_state_workflow_data(self, db_session, sample_cost_agent):
    """Test complex workflow data storage"""
    complex_data = {
        "customer_id": "test-123",
        "analysis": {
            "total_cost": 120000,
            "potential_savings": 60000,
            "recommendations": [
                {"type": "rightsizing", "impact": 30000},
                {"type": "spot_instances", "impact": 30000}
            ]
        },
        "metadata": {
            "started": datetime.utcnow().isoformat(),
            "version": "1.0"
        }
    }

    state = AgentState(
        agent_id=sample_cost_agent.id,
        status="active",
        current_workflow="deep_analysis",
        workflow_data=complex_data,
        resource_locks=[],
        error_count=0,
        last_heartbeat=datetime.utcnow()
    )
    db_session.add(state)
    db_session.commit()

    # Retrieve and verify
    retrieved = db_
```